Alinx AX7035t 开发板实验

——基于埃氏筛法的 6 位素数计算与显示

姜俊彦[1]，刘镇豪[2]，吴尚哲[3]

目录

[1]姜俊彦，中国科学院大学，北京，本次负责
[2]刘镇豪，中国科学院大学，北京，本次负责
[3]吴尚哲，中国科学院大学，北京，本次负责

1 实验题目

素数循环显示: 利用 Alinx AX7035t 开发板，使用 Verilog 编程实现计算并显示 2-999999 之间的素数。6 个七段数码管显示素数，4 个 LED 显示当前模式，4 个按钮选择循环模式

- 按键 1：递增，每秒变一次（上电之后的默认模式）
- 按键 2：递减，每秒变一次
- 按键 3：递增，最快速度
- 按键 4：递减，最快速度

注意事项：

1. 不可以提前将素数计算好存储在 FPGA 内，必须运行时计算
2. 每个小组独立完成，提交一份源码, 一份说明文档，均使用 PDF 电子版提交
3. 组内成员最终共享大作业成绩，不做区别对待

2 实验过程

2.1 实验平台

- 操作系统：Windows 11 专业版 22H2
- 开发平台：Vivado v2023.1(64-bit)
- 硬件平台：XLINX ARTIX-7 系列 AX7035 FPGA 开发平台
- 协作平台：GitHub

2.2 理论调研

2.3 总体规划

2.4 开发日志

3 实验代码

7 附录

7.1 实验源码

```
module top
(
input           clk,
input           rstn,
output [3:0]    led,
input  [3:0]    key,
output [5:0]    seg_sel,
output [7:0]    seg_dig
);
wire [3:0]          key_signal;
wire [3:0]          key_pulse;
wire                rstn_signal;
wire                tick;
wire                one_second;
wire                select;
wire                reset;
wire [47:0]         seg;
wire [19:0]         cnt_20b;
wire [23:0]         cnt_24d;

//1秒计时器
Count_to_one_second timer(clk,one_second);

//按键除抖及脉冲
Killshake Killshake(clk,rstn,rstn_signal);
genvar j;
generate for(j = 0; j < 4; j = j + 1) begin
Killshake Killshake (clk,key[j],key_signal[j]);
Edgedetect Edgedetect (key_signal[j],clk,key_pulse[j]);
end
endgenerate

//Led控制及模式选择
ledcontrol ledcontrol(clk,rstn_signal,key_pulse,led);
modecontrol modecontrol(clk,rstn_signal,one_second,led,key_pulse,reset,select,tick);

//埃氏筛法
isprime solver(clk,reset,tick,select,cnt_20b);

//显示模块
binary_20b_to_bcd_6d transformer(cnt_20b,cnt_24d);
genvar i;
generate for(i=0; i<6; i=i+1) begin
led7seg_decode d(cnt_24d[i*4 +: 4], 1'b1, seg[i*8 +: 8]);
```

```
45    end
46    endgenerate
47    seg_driver #(6) driver(clk, rstn_signal, 6'b111111, seg, seg_sel,
          seg_dig);//数码管驱动，48宽（6*8）数据显示
48
49    endmodule
50
51
52    module ledcontrol
53    (
54    input           clk,
55    input           rstn_signal,
56    input  [3:0]    key_pulse,
57    output [3:0]    led
58    );
59
60    reg    [3:0]    led_r;
61    always @(posedge clk or negedge rstn_signal) begin
62    if(~rstn_signal) begin
63    led_r <= 4'b1110;
64    end
65    else if(~key_pulse[0]) begin
66    led_r <= 4'b1110;
67    end
68    else if(~key_pulse[1]) begin
69    led_r <= 4'b1101;
70    end
71    else if(~key_pulse[2]) begin
72    led_r <= 4'b1011;
73    end
74    else if(~key_pulse[3]) begin
75    led_r <= 4'b0111;
76    end
77    end
78    assign led = led_r;
79    endmodule
80
81
82
83
84    module modecontrol
85    (
86    input           clk,
87    input           rstn_signal,
88    input           one_second,
89    input [3:0]     led,
90    input [3:0]     key_pulse,
91    output          reset,
92    output          select,
93    output          tick
```

```
94  );
95
96  reg                 tick_r;
97  reg                 select_reg;
98  reg  [1:0]          reset_r;
99
100 always @(posedge clk) begin
101 if(~led[0]) begin
102 tick_r <= one_second;
103 select_reg<=1;
104 end
105 else if(~led[1])begin
106 tick_r <= one_second;
107 select_reg<=0;
108 end
109 else if(~led[2])begin
110 tick_r <= 1;
111 select_reg<=1;
112 end
113 else if(~led[3])begin
114 tick_r <= 1;
115 select_reg<=0;
116 end
117 else begin
118 tick_r<=0;
119 end
120 end
121
122
123 always @(posedge clk) begin
124 reset_r<={reset_r[0],(&key_pulse)&rstn_signal};
125 end
126
127 assign tick = tick_r;
128 assign select=select_reg;
129 assign reset=reset_r[1];
130
131 endmodule
132
133 module Edgedetect
134 (
135 input   key,    // 按钮输入
136 input   clk,    // 时钟信号
137 output  pulse   // 脉冲输出
138 );
139
140 reg     key_prev;   // 存储前一个按键状态
141 reg     pulse_reg;  // always块中储存状态
142
143
```

```
144   always @(posedge clk) begin
145   key_prev <= key;
146   // 当检测到按键的负沿时，生成脉冲
147   if (key_prev & ~key)
148   pulse_reg <= 0;
149   else
150   pulse_reg <= 1;
151   end
152
153   assign pulse = pulse_reg;
154
155   endmodule
156
157   module Killshake
158   (
159   input   clk,    // 时钟信号
160   input   key,    // 含噪声的按键输入
161   output  signal  // 清洁的按键输出
162   );
163
164   parameter   DEBOUNCE_TIME = 1000000;    // 去抖时间阈值，根据时钟频率调整,1/50s
165   reg [19:0]  count;                      // 计数器，位宽取决于DEBOUNCE_TIME
166   reg         key_state;                  // 存储稳定后的按键状态
167   reg         signal_reg;                 // always块中储存状态
168
169   always @(posedge clk) begin
170   if (key == key_state) begin
171   // 如果当前输入状态与去抖后的状态相同，则增加计数器
172   if (count < DEBOUNCE_TIME)
173   count <= count + 1;
174   else
175   signal_reg <= key_state; // 更新输出状态
176   end else begin
177   // 如果输入状态改变，重置计数器并更新去抖后的状态
178   count <= 0;
179   key_state <= key;
180   end
181   end
182
183   assign signal = signal_reg;
184
185   endmodule
186
187
188   //数码管驱动
189   module seg_driver #(parameter NPorts=8)
190   (
191   input                   clk, rstn,
192   input [NPorts-1:0]      valid_i,
193   input [NPorts*8-1:0]    seg_i,
```

```
194     output reg [NPorts-1:0]    valid_o,
195     output [7:0]               seg_o
196     );
197
198     reg [14:0]          cnt;        // 15 位寄存器 cnt，用于计数，0<=cnt<=2^15-1
199     reg [NPorts-1:0]    sel;        // NPorts 位（即8位）的寄存器
            sel，用于选择当前输入端口
200     always @(posedge clk or negedge rstn)
201     if(~rstn)
202     cnt <= 0;
203     else
204     cnt <= cnt + 1;
205
206     always @(posedge clk or negedge rstn)
207     if(~rstn)
208     sel <= 0;
209     else if(cnt == 0)
210     sel <= (sel == NPorts - 1) ? 0 : sel + 1; // 若条件(sel == NPorts -
            1)为真，将sel赋值为0，否则sel+1，循环刷新
211
212     always @(sel, valid_i) begin // 使用 sel 和 valid_i 作为敏感信号的 always 块
213     valid_o = {NPorts{1'b1}}; // 初始化 valid_o 为全 1 的向量，表示所有输出端口有效
214     valid_o[sel] = ~valid_i[sel]; //
            取反当前选择的输入端口的有效性，表示相应输出端口的有效性
215     end
216
217     assign seg_o = ~seg_i[sel*8+:8]; //取反从sel_i寄存器索引开始选择的8位数据段，赋值给
            seg_o
218
219     endmodule
220
221     module led7seg_decode
222     (
223     input [3:0] digit,
224     input valid,
225     output reg [7:0] seg
226     );
227     always @(digit)
228     if(valid)
229     case(digit)
230     0: seg = 8'b00111111;//0
231     1: seg = 8'b00000110;//1
232     2: seg = 8'b01011011;//2
233     3: seg = 8'b01001111;//3
234     4: seg = 8'b01100110;//4
235     5: seg = 8'b01101101;//5
236     6: seg = 8'b01111101;//6
237     7: seg = 8'b00000111;//7
238     8: seg = 8'b01111111;//8
239     9: seg = 8'b01101111;//9
```

```
240     default: seg = 0;
241     endcase
242     else seg = 8'd0;
243 
244     endmodule
245 
246 
247     module isprime #(parameter N=999999)
248     (
249     input clk,rstn,tick,
250     input select,
251     output [19:0] cnt_20b
252     );
253     reg [19:0]      w_addr;          //写入的数据的地址
254     reg             w_data;          //写入的数据
255     reg             wea;          //使能端
256     reg [19:0]      r_addr;          //读取的数据的地址
257     wire            r_data;          //读取的数据
258 
259     reg [19:0]      i;               //外层循环变量
260     reg [19:0]      j;               //内层循环变量
261     reg             en;
262     reg             done;
263 
264 
265     reg [19:0]      cnt_temp_reg;
266     reg [19:0]      cnt_20b_reg;
267 
268     reg [2:0]       timer;
269     reg             hold;
270 
271     always @(posedge clk or negedge rstn) begin
272     if(!rstn) begin
273     cnt_temp_reg<=(select)?2:N;
274     cnt_20b_reg<=(select)?2:N;
275     wea<=0;
276     i<=2;
277     j<=0;
278     en<=0;
279     done<=0;
280     timer<=0;
281     hold<=1;
282     end
283     else if (i*i<=N) begin
284     if(en==0)begin
285     r_addr<=i;
286     if(timer>2)begin
287     timer<=0;
288     hold<=0;
289     end
```

```
290     else begin
291     timer<=timer+1;
292     hold<=1;
293     end
294     if(!hold) begin
295     if (r_data==0) begin
296     en<=1;
297     j<=i+i;
298     end
299     else begin
300     i<=i+1;
301     end
302     end
303     end
304     else if(en==1) begin
305     if(j<N)begin
306     wea<=1;
307     w_addr<=j;
308     w_data<=1;
309     j<=j+i;
310     end
311     else begin
312     wea<=0;
313     en<=0;
314     i<=i+1;
315     end
316     end
317     end
318     else begin
319     done<=1;
320     if(done) begin
321     if (((cnt_temp_reg<N)&(select))|((cnt_temp_reg>=2)&(~select))) begin
322     r_addr<=cnt_temp_reg;
323     if(hold) begin
324     if(timer>2)begin
325     timer<=0;
326     hold<=0;
327     end
328     else begin
329     timer<=timer+1;
330     end
331     end
332     else begin
333     if ((~r_data)&tick) begin
334     cnt_20b_reg<=cnt_temp_reg;
335     cnt_temp_reg<=(select)?cnt_temp_reg+1:cnt_temp_reg-1;
336     hold<=1;
337     end
338     else if ((r_data)) begin
339     cnt_temp_reg<=(select)?cnt_temp_reg+1:cnt_temp_reg-1;
```

```
340     hold<=1;
341     end
342     else if((~r_data)&(~tick)) begin
343     cnt_temp_reg<=cnt_temp_reg;
344     end
345     end
346     end
347     end
348     end
349     end
350     assign cnt_20b=cnt_20b_reg;
351     ram_ip ram_ip_inst_1
352     (
353     .clka      (clk          ),     // input clka
354     .wea       (wea          ),     // input [0 : 0] wea
355     .addra     (w_addr       ),     // input [19 : 0] addra
356     .dina      (w_data       ),     // input [0 : 0] dina
357     .clkb      (clk          ),     // input clkb
358     .addrb     (r_addr       ),     // input [19 : 0] addrb
359     .doutb     (r_data       )      // output [0 : 0] doutb
360     );
361     endmodule //isprime
362
363
364
365     module binary_20b_to_bcd_6d #(parameter N = 20,parameter M = 24)
366     (
367     input  [N-1:0]  input_20b,
368     output [M-1:0]  output_6d
369     );
370     reg [3:0]       digits [5:0];
371
372     integer i;
373     always @(input_20b) begin
374     for (i = 0; i < 6; i = i+1) begin
375     digits[i] = 4'd0;
376     end
377     for(i = N-1; i >= 0; i = i-1) begin  //加3移位法
378     if (digits[0] >= 4'b0101) digits[0] = digits[0] + 4'b0011;
379     if (digits[1] >= 4'b0101) digits[1] = digits[1] + 4'b0011;
380     if (digits[2] >= 4'b0101) digits[2] = digits[2] + 4'b0011;
381     if (digits[3] >= 4'b0101) digits[3] = digits[3] + 4'b0011;
382     if (digits[4] >= 4'b0101) digits[4] = digits[4] + 4'b0011;
383     if (digits[5] >= 4'b0101) digits[5] = digits[5] + 4'b0011;
384
385     digits[5][3:0] = {digits[5][2:0], digits[4][3]};
386     digits[4][3:0] = {digits[4][2:0], digits[3][3]};
387     digits[3][3:0] = {digits[3][2:0], digits[2][3]};
388     digits[2][3:0] = {digits[2][2:0], digits[1][3]};
389     digits[1][3:0] = {digits[1][2:0], digits[0][3]};
```

```
    digits[0][3:0] = {digits[0][2:0], input_20b[i]};
    end
    end
    assign output_6d ={digits[5],digits[4],digits[3],digits[2],digits[1],digits[0]};

    endmodule

    module Count_to_one_second #(parameter Count_To = 50_000_000)
    (
    input clk,
    output one_second
    );

    reg [31:0]  counter;
    reg         one_second_r;

    always @(posedge clk) begin
    if((counter < Count_To) && ~one_second_r) begin
    counter <= counter + 1;
    end
    else if((counter < Count_To) && one_second_r) begin
    one_second_r <= 0;
    counter <= counter + 1;
    end
    else begin
    counter <= 0;
    one_second_r <= 1;
    end
    end

    assign one_second = one_second_r;

    endmodule
```

7.2 开发日志

参考文献

[1]